Hayden Miedema and Doug Money

CIS 451

Professor Kurmas

Lab 6: Understanding Pipeline Operations

1. How many cycles does it take to fill the pipeline? (Notice that DLXview labels the cycles beginning with zero. To avoid ambiguity, fill in the blank: "In DLXview, the first cycle during which all the pipeline stages are busy is labeled\_\_\_\_\_?

**Answer: 4, this is when all of the pipeline stages are busy**

1. How many cycles does it take for the computer to execute the first instruction completely?

**Answer: 5 cycles. The add instruction is out of the pipeline during stage 5.**

1. What pipeline stage is the instruction and r6, r7, r8 in during cycle number 5 (i.e., the cycle labeled 5)?

**Answer: Stage MEM**

1. During which cycle does the the processor begin computing the instruction and r12, r13, r14? (Make sure your answer clearly states "the xth cycle" or "the cycle labeled x").

**Answer: 4th cycle. This is when it enters the IF.**

1. During which cycle does the the processor finish computing the instruction and r12, r13,r14? (Make sure your answer clearly states "the xth cycle" or "the cycle labeled x").

**Answer:** **It is completely out of the pipe in the 9th cycle.**

1. Which registers are being read during cycle 5 (the cycle labeled 5)?

**Answer: R13 and R14**

1. Which registers are being written during cycle 5 (the cycle labeled 5)?

**Answer: R3**

1. Where does input to the main ALU come from during cycle 2 (the cycle labeled 2)?

**Answer: The input comes from the ID/EX pipeline, then into the MUX’s before going into the ALU. The value comes from register R2 with offset 0.**

1. Where does input to the main ALU come from during cycle 3 (the cycle labeled 3)?

**Answer: The input again comes from the ID/EX pipeline, then into the MUX’s. The values passed through come from registers R4 and R5.**

1. What is the purpose of all of the **nops** at the end of the sample program?

**Answer: They are used as “dummy” instructions to push the rest of the instructions through the pipeline. NOP is used to prevent hazards from occurring.**

1. How does the DLX pipeline architecture resolve this conflict?

**Answer: The IF and MEM stages in the DLX pipeline are separated. The values are held in registers for those stages until a clock cycle. On a clock cycle, the instructions are carried out and moved out of the IF or MEM stages and move into the next stage of the pipeline.**

1. What purpose does this adder serve?

**Answer: This adder is used for the branch instruction.**

1. Identify all pairs of instructions that have a data dependency. In particular identify all pairs of instructions (not necessarily adjacent) where the result of the second instruction depends directly on the result of the first.

**Answer: sub r3, r4, r5 / and r6, r3, r7**

**sub r3, r4, r5 / or r8, r9, r3**

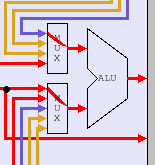
1. Describe in your own words how the DLX hardware addresses this dependency problem. Your answer should be precise enough to convince me that you understand the mechanism used.

**Answer: The DLX pipeline fixes this by storing the value in the pipeline register, which is where the instruction needing a value looks for the given value. Rather than trying to go straight to the register, the pipeline register is used as an intermediate.**

1. Trace through the progress of the fourth instruction in a manner similar to that used in the section above ("**A simple pipelined program**"). There is a "visual bug" or discrepancy that you will notice as you trace through the execution of this program. Identify it.

**Answer: It is not drawing a line through the bottom MUX.**

1. Step to cycle 4. Use KSnapshot (or another tool of your choice) to capture the main DLXview window. Print the snapshot and highlight the set of wires that shows how the result of sub r3, r4, r5 is routed directly to the main ALU.



1. Which value of r3 will be read if the DLX processor used the register file provided for Project 2? If the wrong value would be read, describe how you would properly coordinate the reading and writing of the registers.

**Answer: The old value of r3 will be used. You would coordinate the correct value by adding a mux inside the register to pick the correct value. The mux will use a fwd selector.**

1. Why must the **add** operation be delayed one cycle? Your answer should consider timing issues and functional units. Be sure to explain why forwarding cannot solve the problem.

**Answer: The add operation has to wait for the R1 value to be loaded from the lw instruction.**

1. Describe how hardware can *detect* a load data hazard.

**Answer: When encountering an add, for example, the hardware must look ahead to see if the lw is going to store into one of the registers being used for addition. So, the data hazard section will compare the two instructions, and will perform a stall if the previous instructions needs to use a register that will be written to in a future cycle.**

1. Why is the branch delay slot necessary? In other words, what would go wrong if we removed the nop?

**Answer: The NOP call for the branch delay must be done in order for the bnez check to go through the adder and the isZero circuit. If the delay wasn’t there, there would be an incorrect comparison value.**

1. Suppose you had a smarter compiler. Explain what it could put in the branch delay slot instead of a nop.

**Answer: If we take the last instruction that would be done in the loop and place it where the NOP comes into play there would be no need for a branch delay. This assumes that the bnez call does not depend on the last instruction.**

1. Write an optimized assembly program incorporating your code adjustments. Your solution must still contain a loop.

**Answer: sub r1, r1 0x1**

**addi r4, r4, 0x2**

**beqz r1, LOOP**

**addi r5, r5, 0xa**

**We moved the addi instruction to just after the branch call.**

1. How many cycles does the original program take to assign the final result to register R6? (In other words, during which cycle is the instruction add r6, r4, r5 in the "write-back" phase?)

**Answer: In the 32nd cycle.**

1. How many cycles does your optimized assembly program take to do the same work?

**Answer: In the 27th cycle.**